Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Triple sum

We are given an array $A[0 \dots n-1]$ with n elements, where each element of A is an integer in the range $0 \leq A[i] \leq n$ (the elements are not necessarily distinct). We would like to know if there exist indices $0 \leq i, j, k \leq n-1$ (not necessarily distinct) such that

$$A[i] + A[j] + A[k] = n$$

1. First, consider 2SUM, a simplified version of triple sum where you determine if there exist indices $0 \leq i, j \leq n-1$ (not necessarily distinct) such that

   $$A[i] + A[j] = n$$

   Consider the array $[1, 3, 5]$ and $n = 5$. What are all the possible 2SUMs?

2. Now try encoding the above array into a polynomial to solve 2SUM with one polynomial multiplication. Then, how would you encode an arbitrary array to solve 2SUM?

   Hint: given $p(x) = x^1 + x^3 + x^5$, compute $p(x)^2$. What are the resulting coefficients and exponents in the product? Can they be used to solve 2SUM?

3. Now, design an $\mathcal{O}(n \log n)$ time algorithm for triple sum. Note that you do not need to actually return the indices; just yes or no is enough.

   Food for thought: is it possible to return the number of ways you can add 3 elements from A to equal n?

2 Pattern Matching

Consider the following string matching problem:

Input:

- A string g of length n made of 0s and 1s. Let us call g, the "pattern".
- A string s of length m made of 0s and 1s. Let us call s the "sequence".
- Integer k

Goal: Find the (starting) locations of all length n-substrings of s which match g in at least $n - k$ positions.

Example: Using 0-indexing, if $g = 0111$, $s = 01010110111$, and $k = 1$ your algorithm should output 0, 2, 4, and 7.

(a) Give a $O(nm)$ time algorithm for this problem.

We will now design an $O(m \log m)$ time algorithm for the problem using FFT. Pause a moment here to contemplate how strange this is. *What does matching strings have to do with roots of unity and complex numbers?*

(b) First consider $g = 0110$, $s = 0110$. We know that g and s match at index 0, since all 4 characters of g matches the characters in s.

Now, say we have a function $(x, y) \mapsto f(x, y)$ that returns `len(x)` if the strings x and y match exactly and some number less than `len(x)` otherwise. How can $f(x, y)$ be used to determine the indices where g shows up in s?

(c) Now, lets try to create $f(x, y)$ using the dot product of x and y. That is, $f(x, y) = x \cdot y = \sum_i x[i]y[i]$. Give a counter example for when the dot product returns the wrong answer. Why does this happen?

(d) However, we can modify this dot product to work by first encoding the bits of of x and y. That is, we want to find a bit-mapping function $\Phi : \{0,1\} \mapsto \mathbb{R}$ such that $f(x,y) = \sum_i \Phi(x[i])\Phi(y[i])$ equals to `len(`x`)` if x and y match exactly and some number less than `len(`x`)` otherwise. Construct a mapping Φ for which this property holds.

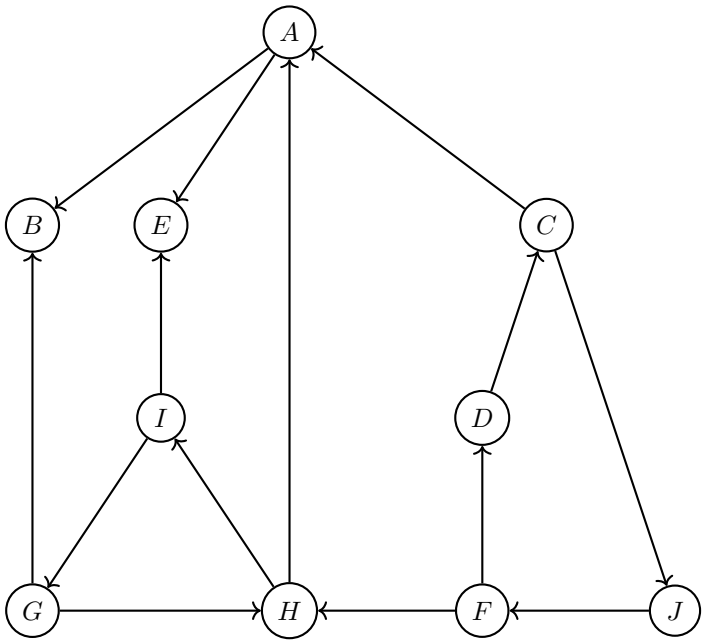
Hint: find Φ *such that*

$$\Phi(x[i])\Phi(y[i]) = \begin{cases} 1 & x[i] = y[i] \\ -1 & x[i] \neq y[i] \end{cases}$$

(e) Now, devise an FFT based algorithm for the problem that runs in time $O(m \log m)$ using the insights of the previous subparts.

Hint: if $p(x) = p_0 + p_1 x + p_2 x^2 + ... + p_{n-1}x^{n-1}$ *and* $q(x) = q_0 + q_1 x + q_2 x^2 + ... + q_{m-1}x^{m-1}$, *then the coefficient of the term* x^b *in* $p(x)q(x)$ *is* $\sum_{i=0}^{b} p_i q_{b-i}$. *Can this be used somehow to compute a dot product?*

3 Graph Traversal



(a) Recall that given a DFS tree, we can classify edges into one of four types:

- Tree edges are edges in the DFS tree,
- Back edges are edges (u, v) not in the DFS tree where v is the ancestor of u in the DFS tree
- Forward edges are edges (u, v) not in the DFS tree where u is the ancestor of v in the DFS tree
- Cross edges are edges (u, v) not in the DFS tree where u is not the ancestor of v, nor is v the ancestor of u.

For the directed graph above, perform DFS starting from vertex A, breaking ties alphabetically. As you go, label each node with its pre- and post-number, and mark each edge as **T**ree, **B**ack, **F**orward or **C**ross.

(b) A strongly connected component (SCC) is defined as a subset of vertices in which there exists a path from each vertex to another vertex. What are the SCCs of the above graph?

(c) Collapse each SCC you found in part (b) into a meta-node, so that you end up with a graph of the SCC meta-nodes. Draw this graph below, and describe its structure.

4 Not So Exciting Scheduling

PNP University requires students to finish all prerequisites for a certain class before taking it; however, they made some mistakes when assigning prerequisites. Thus, some classes at PNP University are NP (Not Possible to take) due to it being impossible to take all its prerequisite classes whilst following the prerequisite rule for them or their prerequisites. Thus, students wish to figure out whether their classes can all be taken or not. Their n classes are labelled with unique identifiers $\{c_1, c_2, \ldots, c_n\}$, and the set of m prerequisites in the form $[c_i, c_j]$ indicate that c_i must be taken before c_j.

Design an algorithm that outputs a potential scheduling of classes (i.e. an order to take all the classes in) if there are no NP classes, return false otherwise.